Literature Review & Research

Introduction

The problem with Current Cyber Security Training Programs

A Critical Analysis of Pre-Existing Cyber Security Games

Find some literature review on turn-based tactics style games if the final project leads that way?

Difficulties that Pre-Existing Cyber Security Games Face

Appropriate Gamification Mechanics

Summary of Requirements Outlined from Literature Review

Summarise the main requirements into a section / table

Why Use a Game - Based Learning Approach?

Resummarise these in a way that's relevant and influenced the decision choice for the final project and **more importantly,** how it relates to the background / problem summary.

Conclusion

The Proposed Final Design

Requirements for the Project

- Adapt this section from future tense to an overall evaluation and justification!

Proposed Idea

Introduction

- The following are requirements for the project in order to meet its goals as a worthy implementation, and research based project..
- **"An analysis and specification of the solution to the problem"**

Functional Requirements

- Add in a charted number reference for these

Non - Functional Requirements

- Add in a charted number reference for these

Constraints

- Add in some content here about some constraints which will make the project particularly difficult, or need to be foregone with a justification

A Brief Account of Work to Date

MoSCoW Analysis

- Move the moscow analysis here

A Justification of this Approach

- Include some alternative approaches here for both applications (tablet/mobile/desktop/web) and server hosting

- Also SQL/Database/account registration if we get to that stage

Design of the Project

Introduction

- Intro about the goals and objectives of said project, the overall design and why

Rules and Mechanics

- Mario Party vs Tactics style explanation
- A flow chart of player choices for different states of the game and how they can interact / present challenge with other players

Visual Design (UIs)

- Include screenshots of wireframes here, or final project

Technical Design (Lobby Matchmaking / Hosting)

- Can talk about master /client server relationships here with a diagram
- High level design of client application / API / other servers

How Gamified Mechanics are Integrated

- As identified earlier and thus meet the requirements!
- This section will probably be quite long with multiple subsections for each mechanic and the value it adds

Cyber Security Teachings

- This section will probably be quite long with a description of types of cyber security attacks/defences identified

Implementation of Project

Development Environment (Unity3D)

Unity3D was chosen as an appropriate development environment because ultimately I wanted to design a game that could be exported to the web which Unity3D has seamless integration for (WebGL). I did consider Phaser3 / HTML as an alternative platform which could progress my web development skills, and integrate this with NodeJS. However, Phaser3 is still a relatively small open source project, whereas Unity3D is a much more widely used industry trade skill – with ample resources and supporting frameworks online.

Furthermore, working with Unity allowed me to hone my understanding of C# which is very syntactically similar to Java, and share a lot of similarities such as being statically typed object-oriented languages; a programming style I am very familiar with now.

Client – Server Multiplayer

Photon Unity Networking 2 (PUN2) is a free to use API which allows developers to host a remote server in which clients can connect to. PUN2 was chosen because it has an ample tutorial base, as well as very strong integration with unity and even support for cross platform (which would make exporting to the web later very easy). PUN2 is typically a paid service, but offers developers free server hosting up to 20 concurrent players, which for the purposes of this research perfect, is more than adequate!

Mirror is an alternative free and open source library which was built to replace Unity's previous deprecated multiplayer libraries (UNET). Although Mirror is completely free to use, relatively simple and highly covered, there were limitations the server and client are **one** project, which would mean that I would either have to run my version of the game constantly to host multiplayer sessions – or develop a server to host game sessions. Thankfully PUN2 makes this process seamless and easily such that other people can participate in my game without my involvement at all.

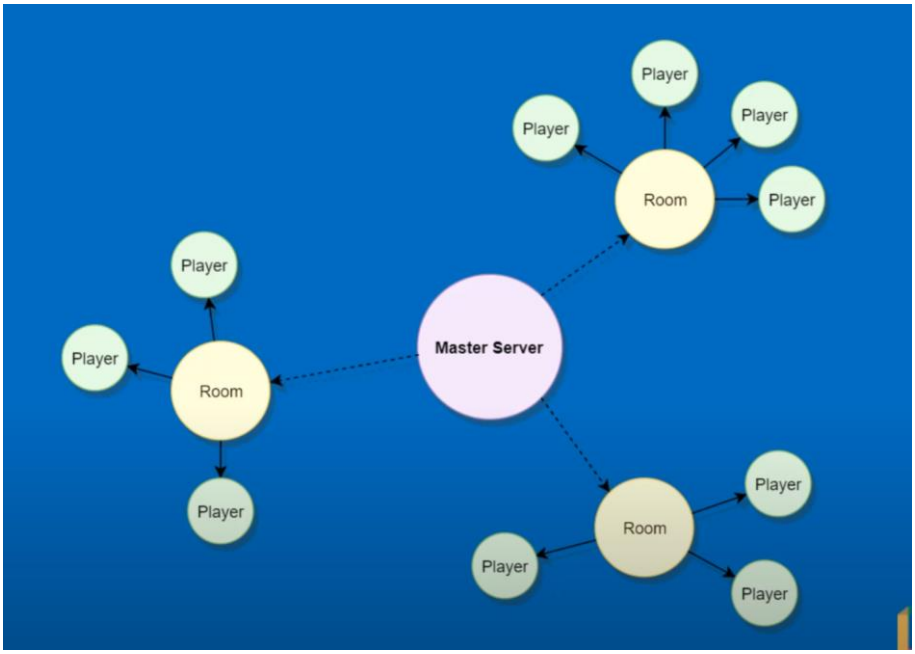
Photon Unity Networking 2 Architecture

Commented [RB1]: In this section, we could reference the master-server architecture and how it fulfils an initial requirement of being a readily available web-based game

PUN2 follows a classic Server – Client architecture; PUN itself hosts cloud servers around the globe in which local clients can connect to remotely. It is free to host 20 concurrent users on any master server, however you'll need to pay to increase that capacity.

When a client connects to the PUN Network initially, it invokes the 'JoinOrCreateRoom()' function which essentially tries to join a non-full room (if it exists), or create one if there are no other rooms (with space) available. The first player (client) to do this becomes the **master client** which is responsible for hosting the game, and for my game, is player 1. If the master client disconnects for any reason, there are options for the next available client (if present) to be promoted to a new master client, however since my developed project is

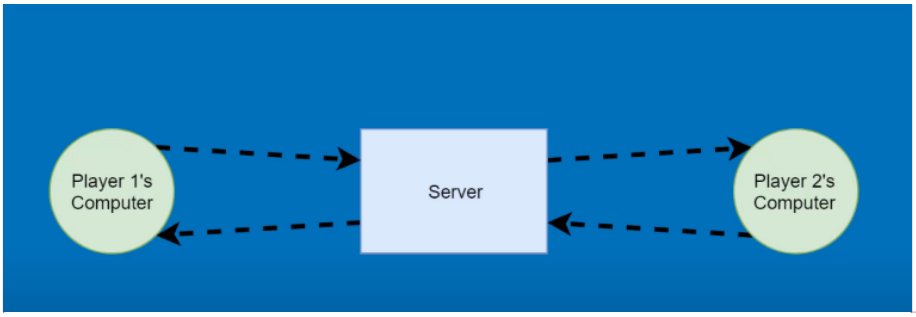
only 2 players, it makes sense just to return the other player to the main menu and end that room session. The biggest number of rooms I can host at one time is therefore 10 (supporting 20 connected clients concurrently).



[Diagram reference: ref – TODO: sketch this myself but for 2 people, but still reference!]

Remote Procedure Calls RPCs

PUN RPCs are a core part of the multiplayer aspect in which a client can communicate the position or state of his game directly to all other clients.



[Insert diagram similar to this, but to illustrate how a client creates and sends an RPC call]

Commented [RB2]: Can include take damage RPC as a direct example of its use

For example, when a new turn is loaded, the client of the previous turn will call photonView.RPC("InitiateNextTurn", RpcTarget.ALL); where the photon view is the ID of the player/client invoking the message, InititateNextTurn is the function the client is telling all other clients to run, and RpcTarget.ALL represents all other connected clients. For larger games, it's possible to pass in the client's particular photon ID here such that you only communicate with one client, and not all.

Furthermore, if the master client who was to begin the game was to invoke photonView.RPC("BeginGame", RpcTarget.Allbuffered); can apply, where buffered instructs the call to wait and ensure **all** clients will receive the message even if they haven't fully loaded the game scene (in the case of delayed connections).

Movement Mechanics – Breadth First Search with Shorted Path

In order to calculate the pathfinding movement possibilities of each unit on the tile map, the player controller inherits a pathfinding class which utilises an implementation of breadth first search. BFS with shorted path is used to calculate all the possible tiles a unit can move to within the maximum movement distance range specified by the Units properties.

At the end of the selected tile, the algorithm returns the shorted path to move to that location tile within range. Essentially, the current tile the unit on is processed first, then each neighbouring node outwards (in all directions) is processed (if applicable) and only processed once – in a breadth first search like manor. [Tutorial here]

1. At the start of runtime, all tiles (nodes) on the board are located and cached to save performance during calculations later on
2. When a unit is selected, a ray is cast from the unit's location to determine the tile/node below it
    - From this tile / node, all neighbours (left, right, up, down) are calculated and added to adjacency list
3. We then initiate breadth first search to find all selectable tiles:
    - We first enqueue the current tile to a Queue of tiles that need to be explored
        - We set this tile visited to true so that we never recheck/come back to this tile
    - We then iterate through the Queue all while it is not empty
        - If this ever reached 0, then we would be in a position where we had no legal moves (surrounding by a block in all 4 movement directions)
        - We add that tile to a list of selectable tiles, and change that colour
        - After processing that tile, we add all neighbours in the adjacency list to the queue for further processing

Commented [RB3]: ▪Note BFS here is used just to find selectable tiles, not a path to that tile

- If a tile in that list has already been marked as visited, we do not this to the queue for processing
- For each tile in this adjacency list, we set the parent node of this tile as the current tile before it, that way we can backtrack a path to the destination later on if needed
- We then mark this child node as visited, as set its distance as 1 + the distance of the previous tile

Commented [RB4]: This distance is important so that we can stop running BFS as soon as this distance is reached as specified by the maximum movement distance of the unit

Testing Strategy and Results

ParallelSync for Quick Builds / Debugging

ParallelSync is a small open source extension for Unity3D which allows you to clone the Unity development environment and run multiple editors of the same project (which means you do not have to build and run the game every time to test 2 or more players!). This extension has proven to be an invaluable timesaver for the development and quick testing of multiplayer development. ParallelSync works by cloning the structure of the project without needing to duplicate and load all files/assets/game states, instead it establishes pointers to the original files/assets and game scenes which have read access only (and do not affect the state of the master development environment). Because of this, it is easy to create, load and destroy as many clone environments as required.

- Need to research multiple methods on how to test an application
- https://www.softwaretestinghelp.com/types-of-software-testing/

JUnit Test Cases

- Write a test case that simulates the walkthrough of client connecting (loading game, taking turn, winning, registering, score updating, achievements being earned etc)
- Other general backend cases, version control is another one! (compatibility)

Integration Testing

- Especially for Networking feature
    - Identify all edge cases and ensure checks are valid for each one (include a state machine with transition!)

UI / Interface Testing Testing

- Testing availability functionality / buttons etc with a table of all edge CASES

Performance Testing

- Since Unity is still a relatively new tool to me, it was important to make a silly mistake with over reliance on any of the networking / continual update hugs
- Profile Analyser
- To review performance analysis and identify performance hogs / what needs to be optimised
- **Loading testing, tho we only have 20 CCU limit anyway**

User Interaction / Feedback Testing / Acceptance Testing

- Double check that this doesn't require ethics approval and is a valid form

- **Usability testing**

Exploratory Testing

- Explore through certain aspects of the application, cleaning up redundant processes, modularising the project as much as possible and keep a track of these

A Critical Evaluation

Evaluation of Approach

- What other alternatives could've been done
- What other APIs / thingymabobs could've been used
- Have all the MoSCoW tasks been achieved?

Evaluation of Final Implementation

- What gamified mechanics have been implemented
    - What mechanics are missing that ideally would've been included given more time
- Does the project meet all the functional and nonfunctional requirements?
- What needs more time or development?
- Is it good on multiple devices as a web application, i.e tablet with touch based response, or just good on a desktop scenario?
- Are there any other features that could have been added to improve the overall complexity of the project
- Comparison to **other online similar projects / prototypes / research**

User Feedback Evaluation

- If time permits, write a pre-analysis questionnaire then get the user to experience the game, then write a post-analysis questionnaire
- Collect some qualitative feedback about overall impressions
- Is the usability aspect good?

Evaluation of Security

- Using PUN vs Mirror has it's security drawbacks, is the application easy to break and cheat? Though this shouldn't be an issue given the light hearted design, like no one would bend heaven and earth to cheat in monopoly?

Evaluation of Testing Strategies

- Are they enough / sufficient?
- Are there any other testing strategies that could've been done given more time?

Conclusions and Future Work

Project Management

- Which tools were the most important
    - What could've been used instead / done differently
- Which techniques were the most important
    - What could have been changed

Reflection

- Risk Assessment
    - Any extra risks / unforeseen stuff / changes?
- Gantt Charts
    - Overall structure / progress? Start anything earlier? More time for exams!!!

Future Additions

Conclusion

-

Bibliography

Appendices